You know the Science.
Do you know your Code?

Automated Code Analysis and Transformation tools to Support Scientific Computing

Semantic Designs, Inc.

Ira Baxter

www.semanticdesigns.com

March, 2010

Typical Scientific Code Development

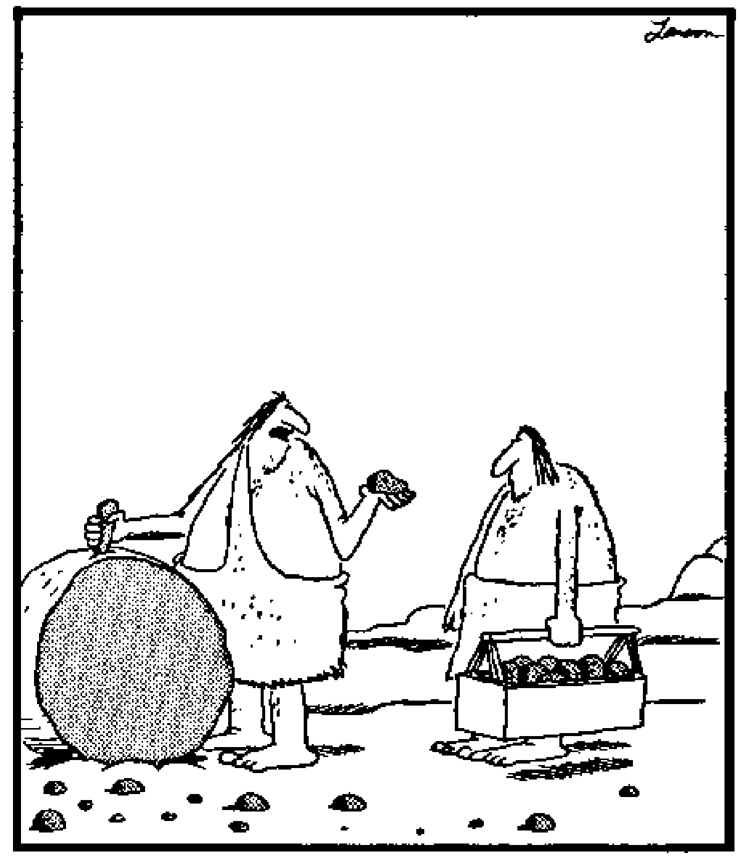
- Choose physics modelling problem
- Determine equations
- Find approximate solvers
- Code/Revise FORTRAN
- Test FORTRAN
- Tune FORTRAN

} Iterate

Software Tools for Scientific Computing

- Text Editor
- Fortran Compiler
- Debugger

- That's it?



"So what's this? I asked for a *hammer*! A hammer! *This* is a crescent wrench! ... Well, maybe it's a hammer. ... Damn these stone tools."

What do we really want?

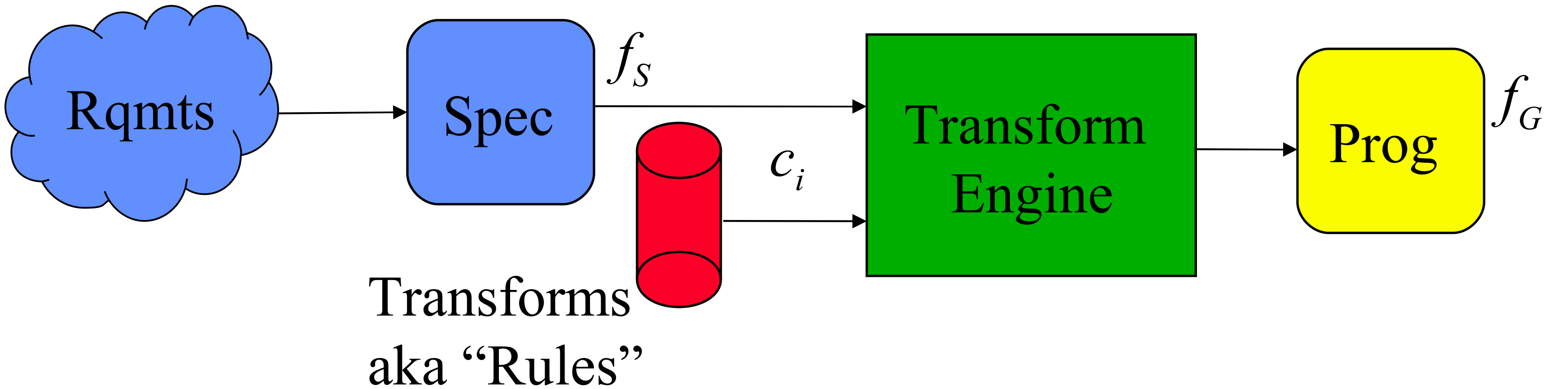
- A way to write a spec in a succinct notation
- Incremental conversion of spec into code
- Capture of rationale for each step
- Means to add implementation knowledge
- Means to revise spec and get revised code

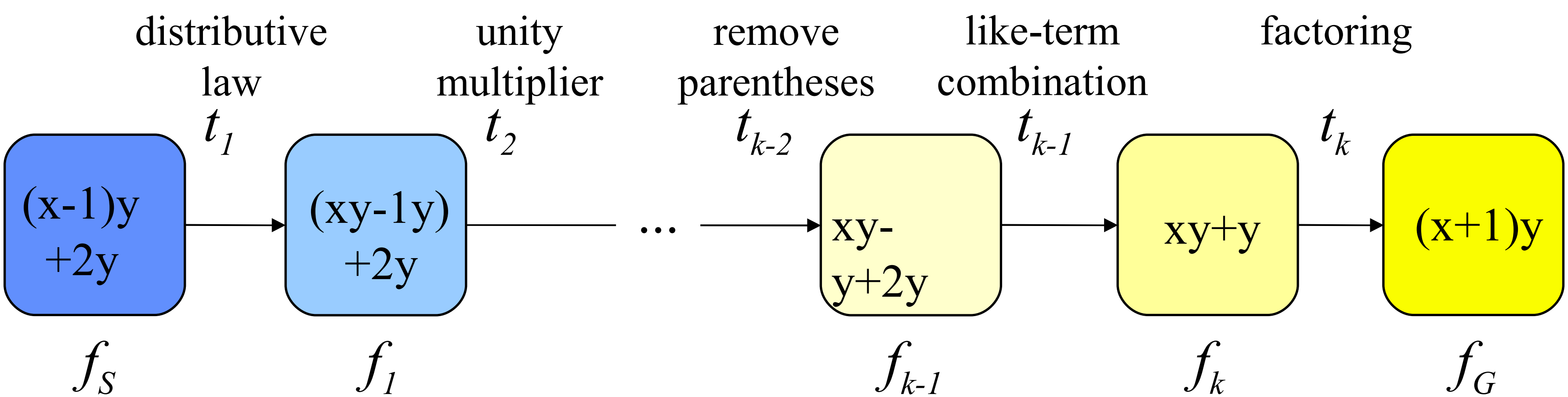
These tools exist!

- Sinapse
  - Financial differential equations → code *www.scicomp.com*
- DMS Software Reengineering Toolkit
  - Arbitrary program transforms *www.semanticdesigns.com*

Key Technology: Transformation Systems

Stepwise Semiautomatic Conversion of Specs to Code





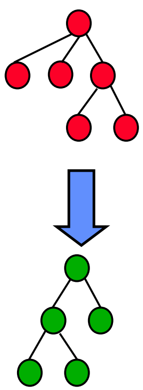
Optimization transform in DMS Rewrite Rule Language

```
default base domain C;

rule use-auto-increment(v: lvalue):
          statement -> statement
   "\v = \v +1"
   rewrites to
   "\v++"
   if no_side_effects(v);
```

Domain Name

Domain Syntax

Rule Condition



Before: `(*Z)[a>>2] = (*Z)[a>>2]+1;`

After: `(*Z)[a>>2]++;`

So what's hard about these tools?

Knowledge Capture

- Defining notations (differential equations, FORTRAN)
- Notation parsers (MATLAB, C++ front ends)
- <u>Computing inferences</u> (symbol properties, information flows)
- <u>Mappings</u> (partial functions) from one notation to another
- Capturing sequence of transformations
- Replay of transformation sequences

Limits what we can do now

- But <u>enables</u> many useful software engineering tasks

USAF B2 Bomber: Automated Legacy Migration

- *Thousands of rules*
- *100% conversion*
- *Reused for F-16 migration*

Automated JOVIAL to C Migration

- Problem: aging 16-bit 1750 microprocessors in B2 Bomber
  - 350,000 lines of mission software in JOVIAL — *Spec*
  - Desperately need more memory space and speed
    - Application functionality enhancements pushing boundaries
  - No deep institutional knowledge about code details

- Solution: DMS + Semantic Designs' services
  - 12 months to implement JOVIAL translator — *Transforms*
    - Uses DMS source-to-source transformation rules
  - 100% automated translation (some minor input edits)
  - Passes ground simulator for B2
  - Staging for installation in aircraft *now*

1M Transformations

Refinement transforms

Jovial to C

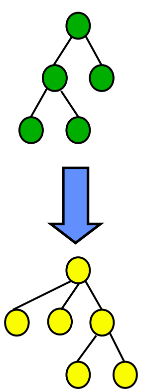
```
default source domain Jovial;
default target domain C;

private rule refine_data_reference_dereference_NAME
                (n1:identifier@C,n2:identifier@C)
                   :data_reference->expression
  = "\n1\:NAME @ \n2\:NAME" -> "\n2->\n1".

private rule refine_for_loop_letter_2
               (lc:identifier@C,f1:expression@C,
                f2:expression@C,s:statement@C)
                     :statement->statement
  = "FOR \lc\:loop_control :
        \f1\:formula BY \f2\:formula; \s\:statement"
    ->
       "{ int \lc = (\f1);
          for(;;\lc += (\f2)) { \s } }"
        if is_letter_identifier(lc).
```

Domain Name

Pattern Variables

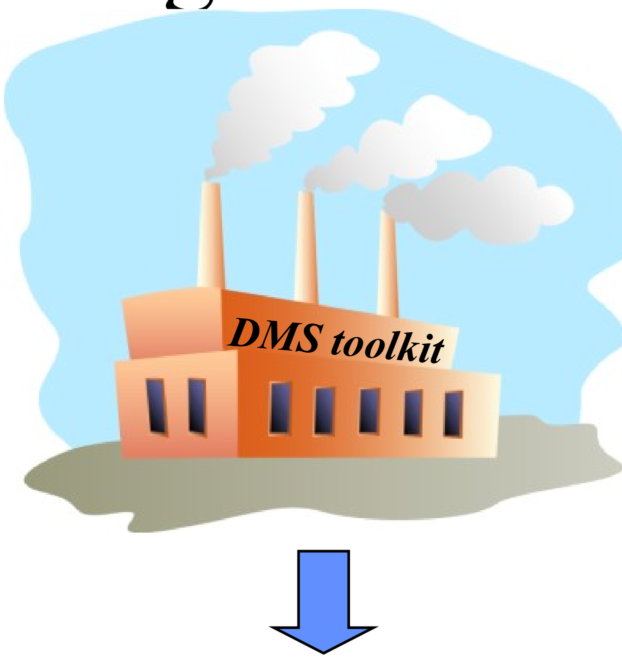
Source Domain Syntax

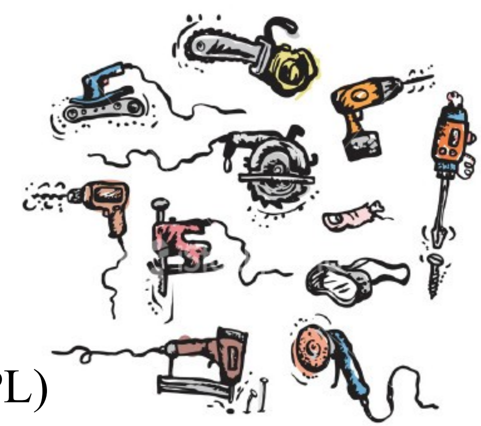
Target Domain Syntax



DMS Software Reengineering Toolkit

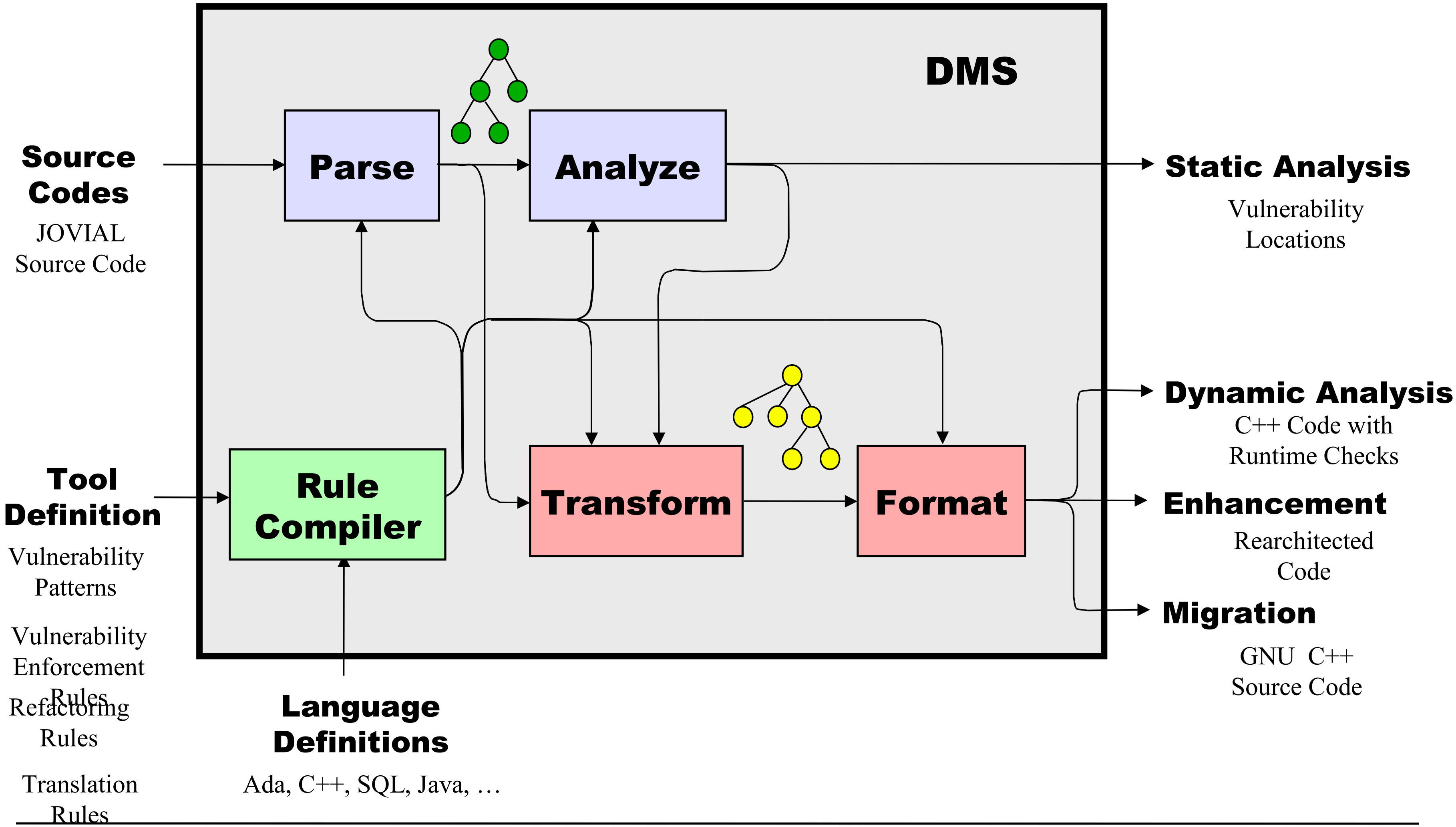
- Metaprogramming machinery
  - Source code analysis + *modification*
- Enables variety of automated SE tasks
- Commercial applications
  - Formatters, Hyperlinked Source Browsers
  - IP protection by code obfuscation
  - Documentation extraction
  - Metrics
  - Preprocessor conditional simplification
  - Test Coverage and Profiling tools
  - Clone Detection and removal
  - *DSL code generation: Factory Automation*
  - *Migrations (JOVIAL to C, C++ to C#)*
  - *Large-scale C++ component restructuring*
  - *SIMD vector generation from data-parallel C++ code*
- Research applications
  - Generic Aspect-weaving (U. Alabama Birmingham)
  - Code generation/quality checking for spacecraft (NASA/JPL)
  - Architecture Extraction (SD)





How DMS Works

Generalized Compiler Technology Specialized to Desired Task



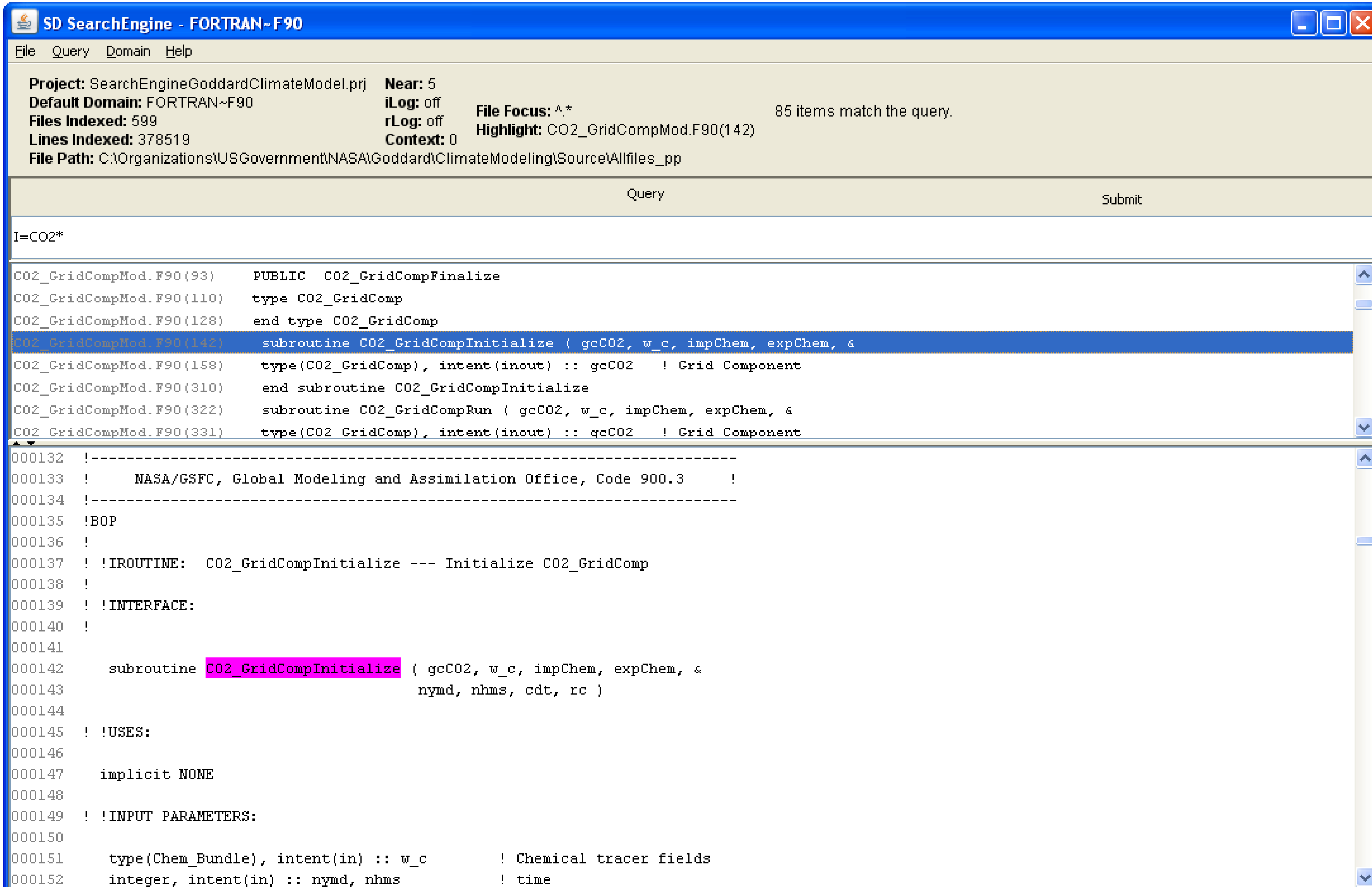
Some (potential) applications *of DMS* for Scientific Computing

- Search Large Application Codes
- Static Analysis: Finding Duplicated Code
- Dynamic Analysis: Test Coverage
- Smart Differencing
- Minimizing re-testing
- Acquiring regression Tests
- SIMD Code Generation from C++
- Physical Units Checking

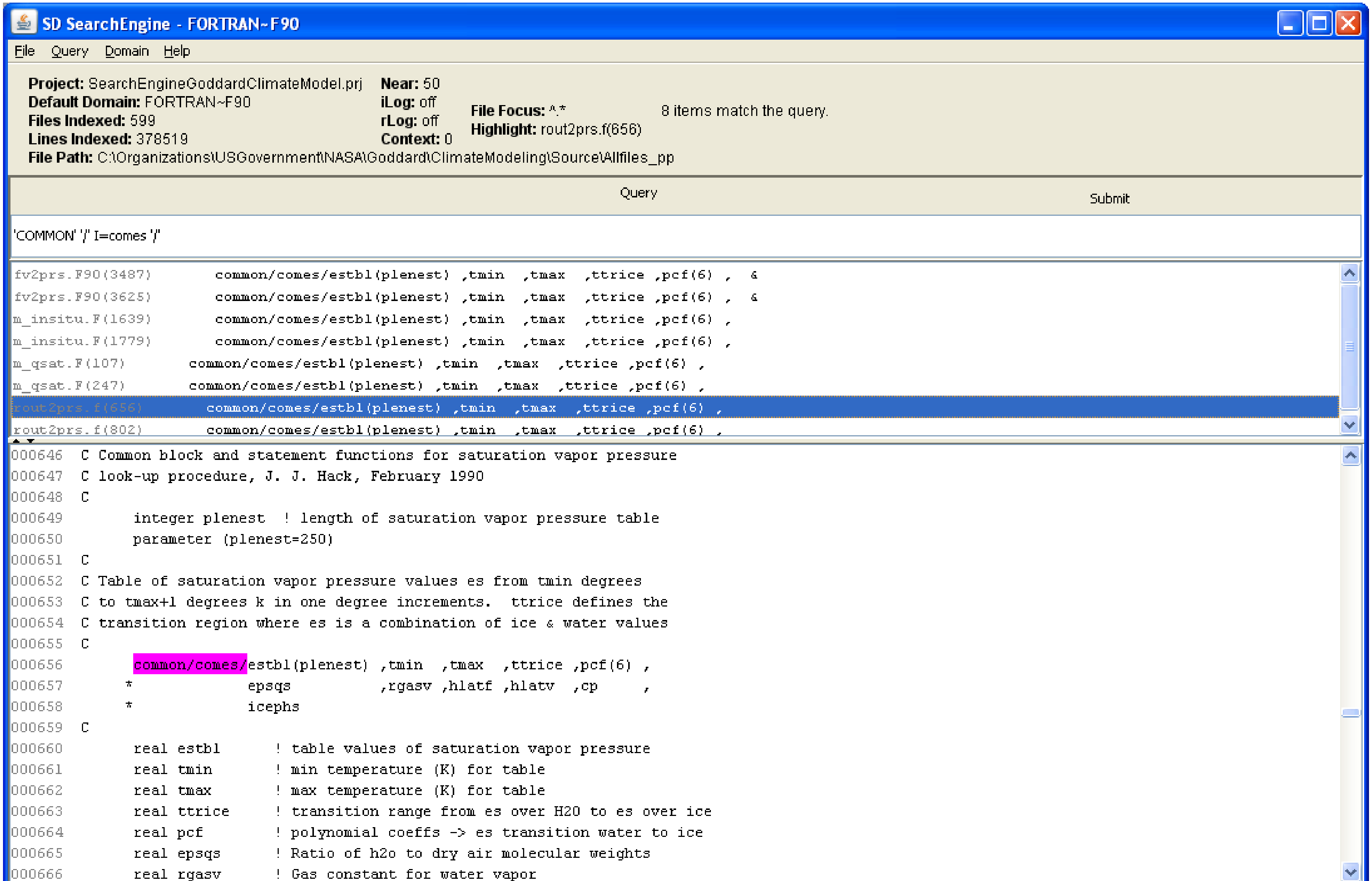
Searching Large Applications

- Some Scientific Codes are huge (1M SLOC, 1000 files)
- Programmers spend 50% of their time looking at code
- One problem: how to find anything?
  - *Solution*: Code Search Engine
    - Fast find across large code bases: C, C++, Fortran
    - Instant display of hits and matching source code

Search: Where's the Subroutine?



Search: Where’s the COMMON block?



Static Code Quality Analysis: Clone Detection

- *Solution*: Find copy/paste/edit duplicated code:
  - Detect Exact and Near Miss hits
- “I fixed this code” (are there other copies?)
  - 20% clones → 20% chance there is other code to fix!
  - What does it cost to *miss* a (cloned) fix?
- Stolen abstractions → should be library routines
- Inconsistent parameters → buggy clones

Clone Detection on Large Fortran Code

FORTRAN~F90 CloneDR (TM) Clone Detector and Reporter, Version 2.2.99

Semantic Designs, Inc.
13171 Pond Springs Road
Austin, TX 78729–7102
+1 512–250–1018
www.semanticdesigns.com

Clone Detection Report for FORTRAN~F90

Project File: C:/Organizations/USGovernment/NASA/Goddard/ClimateModeling/CloneDetectionGoddardClimateModel.prj

Table of Contents

Clone Detection Parameters

Name	Value
Similarity Threshold	95%
Maximum parameter count	5
Minimum Mass (Lines)	3.0
Characters per node	16
Starting height	2

Clone Detector Statistics

Statistic	Value
File Count	599
Total Source Lines of Code (SLOC)	378186
Estimated SLOC before preprocessing	377890
Expanded SLOC after preprocessing	370592
Total clone tuples	4862
Exact-match clone tuples	1847
Near-miss clone tuples	3015
Number of cloned SLOC	130873
Estimated removable SLOC	73362
Possible SLOC reduction %	19.4%
Possible SLOC reduction in expanded file %	19.8%

A copy/paste/edit Clone

CloneTuple388

Back to Main Report

Detected Clone Tuple				
Tuple Mass	Number of Clones in Tuple	Number of Parameters	Similarity of Clones in Tuple	Syntax Category [Sequence Length]
15	3	2	0.961	block_do_construct

Clone Abstraction | Parameter Bindings

Clone Instance (Click to see clone)	Line Count	Source Line	Source File
1	15	765	C:/Organizations/USGovernment/NASA/Goddard/ClimateModeling/Source/Allfiles_pp/decompmodule.F90
2	15	1109	C:/Organizations/USGovernment/NASA/Goddard/ClimateModeling/Source/Allfiles_pp/decompmodule.F90
3	15	953	C:/Organizations/USGovernment/NASA/Goddard/ClimateModeling/Source/Allfiles_pp/decompmodule.F90

Clone Instance	Line Count:	Source Line:	File:
1	15	765	C:/Organizations/USGovernment/NASA/Goddard/ClimateModeling/Source/Allfiles_pp/decompmodule.F90

```
          DO n = 1,zdist(k)
            counter3 = counter2
            DO m = 1,ydist(j)

!     Since this is a regular distribution the definition of
!     tags is dictated by Xdist(I), and appears Ydist(J) times


              l = l+1
              decomp%head(truepe)%starttags(l) = counter3+1
              decomp%head(truepe)%endtags(l) = counter3+xdist(i)
              counter3 = counter3+sizex
            END DO
            counter2 = counter2+sizex*sizey
          END DO
```

The Clone Abstraction

Clone Abstraction	Number of Parameters: 2

```
DO n = 1,zdist(k)
   [[#variable822c36a0]]= [[#variable822c3640]]
  DO m = 1,ydist(j)
    !     Since this is a regular distribution the definition of
    !     tags is dictated by Xdist(I), and appears Ydist(J) times
    l = l+1
    decomp%head(truepe)%starttags(l) = [[#variable822c36a0]]+1
    decomp%head(truepe)%endtags(l) = [[#variable822c36a0]]+xdist(i)
     [[#variable822c36a0]]= [[#variable822c36a0]]+sizex
  END DO
   [[#variable822c3640]]= [[#variable822c3640]]+sizex*sizey
END DO
```

Parameter Bindings			
Parameter Index	Clone Instance	Parameter Code	Value
1	1	[[#822c36a0]]	counter3
1	2	[[#822c36a0]]	counter4
1	3	[[#822c36a0]]	counter3
2	1	[[#822c3640]]	counter2
2	2	[[#822c3640]]	counter3
2	3	[[#822c3640]]	counter2

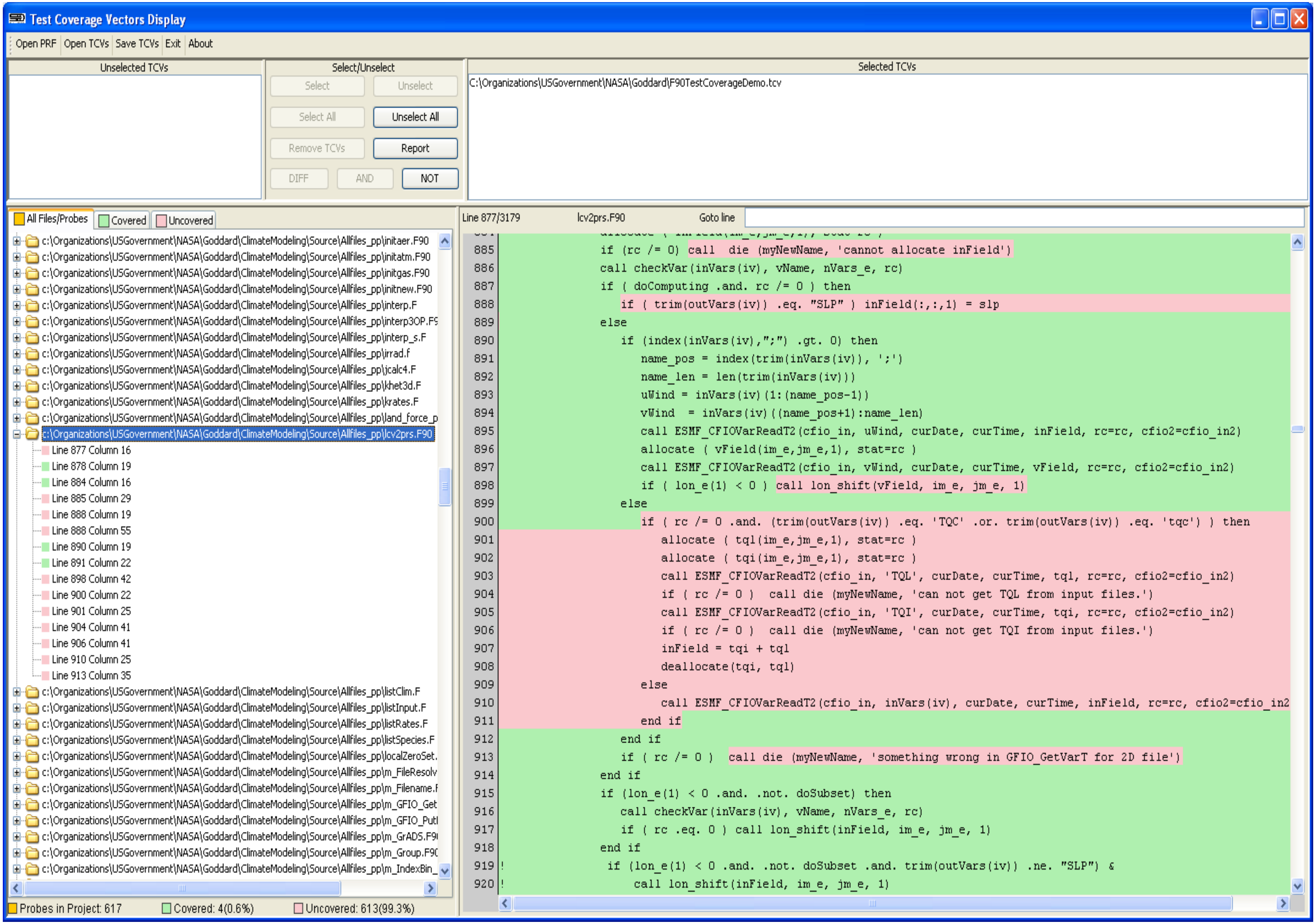
Code sure to cause a bug → wastes Scientist's time

Detected Clone Tuples (Sorted by Total Mass of Tuple)					
Tuple Details	Tuple Mass	Number of Clones in Tuple	Number of Parameters	Similarity of Clones in Tuple	Syntax Category [Sequence Length]
xCloneTuple1	291	31	0	1.000	execution_part_construct_list
xCloneTuple2	679	6	0	1.000	program_unit[7]
xCloneTuple3	1081	4	0	1.000	program_unit[9]
xCloneTuple4	221	14	4	0.982	specification_part
xCloneTuple5	1106	2	3	1.000	subroutine_subprogram[13]
xCloneTuple6	17	40	3	0.963	declaration_construct_list
xCloneTuple7	257	4	4	0.990	execution_part_construct_list
xCloneTuple8	750	2	4	0.996	internal_subprogram_part
xCloneTuple9	674	2	0	1.000	subroutine_subprogram
xCloneTuple10	643	2	4	0.993	subroutine_subprogram[12]
xCloneTuple11	39	16	2	0.968	declaration_construct_list
xCloneTuple12	562	2	0	1.000	program_unit[4]
xCloneTuple13	182	4	2	0.997	subroutine_subprogram
xCloneTuple14	107	6	3	0.994	program_unit[2]
xCloneTuple15	106	6	1	0.999	execution_part_construct_list
xCloneTuple16	514	2	1	0.991	program_unit[4]
xCloneTuple17	114	5	1	0.977	declaration_construct_list
xCloneTuple18	460	2	3	0.997	subroutine_subprogram[2]
xCloneTuple19	16	27	1	0.973	declaration_construct_list
xCloneTuple20	431	2	5	0.961	block_do_construct
xCloneTuple21	143	4	0	1.000	subroutine_subprogram
xCloneTuple22	377	2	2	0.999	subroutine_subprogram[5]
xCloneTuple23	126	4	1	0.997	subroutine_subprogram
xCloneTuple24	75	6	3	0.966	declaration_construct_list
xCloneTuple25	63	6	4	0.973	declaration_construct_list
xCloneTuple26	350	2	0	1.000	subroutine_subprogram[5]

Dynamic Analysis: Test Coverage for F90

- Have you done adequate testing?
- Passes "all my tests" isn't enough
- What about code not exercised?
- Solution: Track executed code
  - Display in UI
  - Produce reports on coverage

Test Coverage for F90 (Mockup)



Reviewing code changes

- Use (classic line-oriented) Diff(?)
  - Advantages: widely available, easily understood
  - Disadvantages:
    - Line granuality: fine detail in statements hard to see
    - Doesn't understand code structure
      - Programmers change constants, identifiers, expressions, statements, block
- Use Smart Differencer(!)
  - Advantages:
    - Detects changes in code structures
    - Reports changes as "rename", copy, delete, move, …
    - Finer grain output → focused reviewing
    - Faster reviews

Diff vs. SmartDiff: misspelled Julian Dates

Diff output: 141 lines

```
! Conversions to and from Julienne dates and find day_of_the_week.
---> ! Conversions to and from Julian dates and find day_of_the_week.
19,20c19,20< !     julienne :: Integer, Optional
< !              If present the julienne day for which the weekday is
---> !     julian :: Integer, Optional
> !              If present the julian day for which the weekday is
31c31< !                            -1=invalid Julienne day
---> !                            -1=invalid Julian day
34c34<       subroutine day_of_week(julienne, weekday, day, ierr)
--->       subroutine day_of_week(julian, weekday, day, ierr)
37c37<       integer,intent(in),optional  :: julienne
--->       integer,intent(in),optional  :: julian
44,45c44,45<       if(present(julienne)) then
<         if(julienne < 0) then
--->       if(present(julian)) then
>         if(julian < 0) then
49c49<         iweekday = mod(julienne+1, 7)
--->        iweekday = mod(julian+1, 7)
51c51<         iweekday = date_to_julienne(ierr=ierr)
--->        iweekday = date_to_julian(ierr=ierr)
82c82< ! Convert a Julienne day to a day/month/year
---> ! Convert a Julian day to a day/month/year
85,86c85,86< !     julienne :: Integer
< !              The julienne day to convert
---> !     julian :: Integer
> !              The julian day to convert
100c100< !                            -1=invalid Julienne day
---> !                            -1=invalid Julian day
103c103<       subroutine
julienne_to_date(julienne,day,month,year,values,ierr)
```

Another 100 lines…

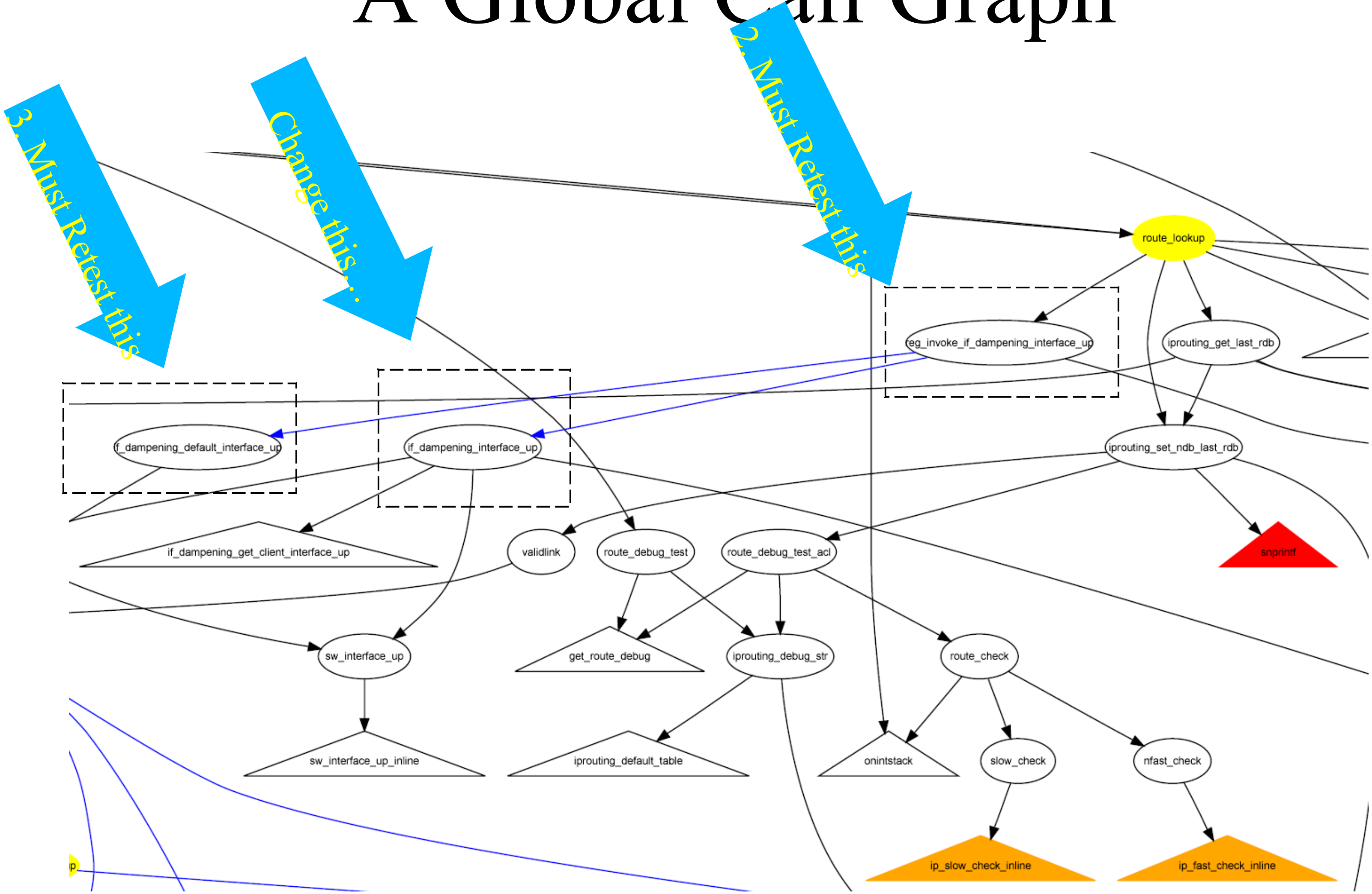
Smartdiff output: 1 line

```
Rename 34.7-265.36 to 34.7-267.34 with
'date_to_julienne'->'date_to_julian',
'julienne_to_date'->'julian_to_date',
 and 'julienne'->'julian'
```

After change, what to Unit retest?

- Modules X directly called by Unit tests
  - Sort of easy to detect with diff
    - Fails badly on moved code
    - Forces retest of modules with just renames
  - Use SmartDiff
- Modules Y that call changed X
  - Need global call graph
  - Use C? Needs points-to analysis
- Modules Z called by changed Y
  - after call to X … if Z uses result from X
  - Need control flow and data flow analysis

A Global Call Graph



Where to get Unit (Regression) Tests?

- Most development have very few (intentions don't count!)
  - Very hard to construct with large, running application
- Running code a possible source
  - Instrument each function
  - Collect argument/result values at runtime
    - Must include all variables *read* by function
  - Generate Unit tests for argument/result pairs
  - Puppetize (modify) code to force enable Unit test execution
- Needs FORTRAN…
  - Control/data flow analysis
  - Transformation to insert instrumentation/puppet code

Compiling C++ for SMP/SIMD machines

- Problem
  - Suddenly, SMPs with SIMD are cheap
    - Variety of targets: PowerPC, Cell, X86, custom CPUs
  - How to get high performance C++ applications running there?

- Solution: Vector C++
  - New vector datatype **`V[ i:j ; m:n ; x:y ]`**
    - Arbitrary dimension array of arbitrary subtype
    - Breaks C++ storage layout rules → enables communication optimizations
  - Data layout specifications
  - Array slices and data parallel operation on (sub)arrays
  - Partial order computations
- Where to get compiler?
  - Use DMS + program transformations
  - Translate VectorC++ to target-specific C++/SIMD operators

SIMD: Prototype VC++

- Robust VectorCpp.h (raw vector implementation as templates)
  - Can enable "Out of range" errors on subscripts (via C++ asserts)
- Dynamic vectors now usable
  - Initial sizing, access/update, parameter passing
- Simple casts between arrays and vectors
- RHS Vector Slicing; some LHS vector slicing
- Elementwise built-in operators on vectors: `+ - * / < > =`
- Elementwise user-defined operations on vectors (a.k.a. lifted functions)
- Dot product `{.}` and Matrix Multiply `{+@*}` implemented
- **forceinline** works for non-lifted functions (but not for methods)
- Reduction of many VC++ operators to **forall** loops containing nested **if**s
  - Feeds into VMX vectorization with slices in inner forall loops
- Fusing of some forall loops (current implementation: not always safe)
  - Fused loop bodies provide better vectorization opportunities

- Compiles to Vanilla C++
- Compiles to C++ with PowerPC VMX SIMD

SIMD: Example Translation to Vanilla C++

VC++

Note scalar inside loop; can't vectorize like this

```
int main () {
    int x[3:7];
    int y[3:7];
    forall (int z=4:6)
        { int p;
          p= x[z]+1;
          x[z]=p*2;
        }
    forall (int z=4:6)
      y[z]=0;
}
```

Note two forall loops fused, will enable better vector machine code generation

C++ translation

```
#include "VectorCpp.h"
int main()
  {
    VectorCpp::Vector<int, …<…, 3, 7> > x;
    VectorCpp::Vector<int, …<…, 3, 7> > y;
    {
      VectorCpp::Vector<int, …<…, 4, 6> > p;
      {
        for (int z = 4; z < 6; z++)
          {
            {
              p[z] = x[z] + 1;
              x[z] = p[z] * 2;
            }
            y[z] = 0;
          }
      }
    }
  }
```

Note p has been lifted out of forall loop and vectorized so that loop can be implemented with vector instructions

SIMD: Example Translation to SIMD Vectors

Pre and post loops handle fractional vectors

VC++

```
int main () {
    int j=2; int m[0:10];
    float x[0:10],y[0:10];
    forall(int q=0:10)
      {  x[q]=x[q]+.3f*y[j];
         if (y[q]!=3.f & x[q]>1.f )
      { if (m[q]==3)
          x[q]=0.0;
      }
       }
    }
```

C++ translation (edited)

```
const float L15[4] = {.3e0f,.3e0f,.3e0f,.3e0f};
#include "VectorCpp.h"
int main() {
  int j = 2; VectorCpp::Vector<int, …, 0, 10> > m;
  VectorCpp::…float…0, 10> > x; VectorCpp::…float…0, 10> > y;
  { int lt11 = 0; int ut12 = 10; // preloop: handle fractional vector
    for (int q=lt11; q<(lt11%4 … ); q++)
      { x[q] = x[q] + .3f * y[j];
        if (y[q] != 3.f & x[q] > 1.)
      if (m[q] == 3)
        x[q] = 0.0; }
    for (int q=lt11%4==0?lt11:lt11+4-lt11%4; q<ut12-ut12%4;
             q += 4)
     { __stvx(__vmaddfp(__lvx(&L15,0),
                  __vspltw(__lvlx(&y[j],0),0),
                  __lvx(&x[q],0)),&x[q],0);
       { _vector4 t13 = __vand(__vcmpequwR(
             __vcmpequw(__lvx(&y[q],0),__vcfux(__vspltisw(3),0)),
                                  __vspltisw(0), 1),
                __vcmpgtfpR(__lvx(&x[q],0),
                        __vcfux(__vspltisw(1),0),1)));
         if (XMComparisonAnyTrue(1))
      {  vector4 t14 = __vcmpequwR(__lvx(&m[q],0),
                                 __vspltisw(3), 1);
              t14 = __vcmpequwR(t13,__vand(t14, t13), 1);
          if (XMComparisonAnyTrue(1))
                 __stvx(__vsel(__lvx(&x[q],0),
                              __vspltisw(0),t14),&x[q],0);
      }}}
      // postloop
      for (int q = ut12 - ut12 % 4; q < ut12; q++)
      { … } } }
```

Note multiply-and-accumulate optimizatons

Note conditional update of x[q] based on nested conditionals in loop

Physical Units Checking
Does your computation make sense?

- Problem: easy to get units wrong in formula

  SNOW_FEET = SNOW_MASS / 12.0

  – Formula complexity, abbreviated names contribute

    SNOW_DEPTH = SNM / 12.0

- *Solution*: Automate Units Checking
  – Annotate code with units
    - No change in performance

  – Tool checks for sensible usage

Physical Units Checking

```
REAL PRECIP_RATE ! u_gram/u_second
REAL DURATION  ! u_hour
REAL SNOW_DEPTH ! u_inches
…
```

Programmer annotations

Implied units

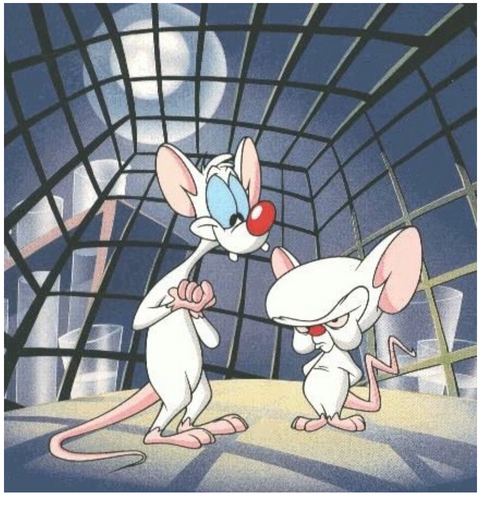
```
SNOW_MASS = PRECIP_RATE * DURATION *
     &          ( 60*60 * u_second / u_hour)
…
SNOW_DEPTH = SNOW_MASS / (12.0 * u_inches/ u_foot)
```

Programmer annotations

Detected error

Other Possible Tools

- Code Refactoring
- Code optimization for parallel machines
- Holy Grail: Differential equations → code



Conclusion

- Big codes are hard to build
- *Many tools* can improve development
- General mass analysis/change tools
  make it practical to get *many tools*
- *DMS is one of these engines*
- *SD already has a number of useful tools*

DMS Toolkit Components

- Parser/PrettyPrinter
  - Multimode lexing and GLR parsing
  - Automatic AST construction from grammar
  - Preprocessor parsing, comment retention
- Existing front ends
  - C, C++, C#, Java, Pascal, Visual Basic
  - COBOL, JCL, FORTRAN, Ada
  - HTML, XML, CORBA IDL
  - Verilog, VHDL
- Procedural API to ASTs
  - Traditional compiler API
- Analysis support
  - Parallel attribute evaluator over AST
  - General symbol table manager
  - ***Control flow graph construction***
  - ***Data flow analysis framework***
  - ***Points-to analysis framework***

- RSL: Transformation Rule Language
  - Patterns in DSL syntax
  - Patterns, rewrites, rewrite rule sets
- Term rewriting engine
  - Associative/Commutative rewrites
  - Constant folding on basic types
- PARLANSE: DMS Procedural Programming Language
  - Custom analyzers/transforms
  - SMP Parallelism
  - Robust exception handling